Problem set 4

2025-10-05

In the next problem set, we plan to explore the relationship between COVID-19 death rates and vaccination rates across US states by visually examining their correlation. This analysis will involve gathering COVID-19 related data from the CDC's API and then extensively processing it to merge the various datasets. Since the population sizes of states vary significantly, we will focus on comparing rates rather than absolute numbers. To facilitate this, we will also source population data from the US Census to accurately calculate these rates.

In this problem set we will learn how to extract and wrangle data from the data US Census and CDC APIs.

1. Get an API key from the US Census at https://api.census.gov/data/key_signup.html. You can't share this public key. But your code has to run on a TFs computer. Assume the TF will have a file in their working directory named `census-key.R` with the following one line of code:

```
census_key <- "A_CENSUS_KEY_THAT_WORKS"
```

Write a first line of code for your problem set that defines `census_key` by running the code in the file `census-key.R`.

```r
## Your code here
census_key <- source("census-key.R")
```

2. The US Census API User Guide provides details on how to leverage this valuable resource. We are interested in vintage population estimates for years 2021 and 2022. From the documentation we find that the *endpoint* is:

```r
url <- "https://api.census.gov/data/2021/pep/population"
```

Use the **httr2** package to construct the following GET request.

```
https://api.census.gov/data/2021/pep/population?get=POP_2020,POP_2021,NAME&for=state:*&key=Y(
```

Create an object called `request` of class `httr2_request` with this URL as an endpoint. Hint: Print out `request` to check that the URL matches what we want.

```
library(httr2)
request <- request(url) |>
  req_url_query(
    get = "POP_2020,POP_2021,NAME",
    `for` = "state:*",
    key = census_key$value
  )
```

3. Make a request to the US Census API using the `request` object. Save the response to and object named `response`. Check the response status of your request and make sure it was successful. You can learn about *status codes* here.

```
response <- request |>
  req_perform()

response$status_code # successful code is 200
```

```
[1] 200
```

4. Use a function from the **httr2** package to determine the content type of your response.

```
# Your code here
resp_content_type(response)
```

```
[1] "application/json"
```

5. Use just one line of code and one function to extract the data into a matrix. Hints: 1) Use the `resp_body_json` function. 2) The first row of the matrix will be the variable names and this OK as we will fix in the next exercise.

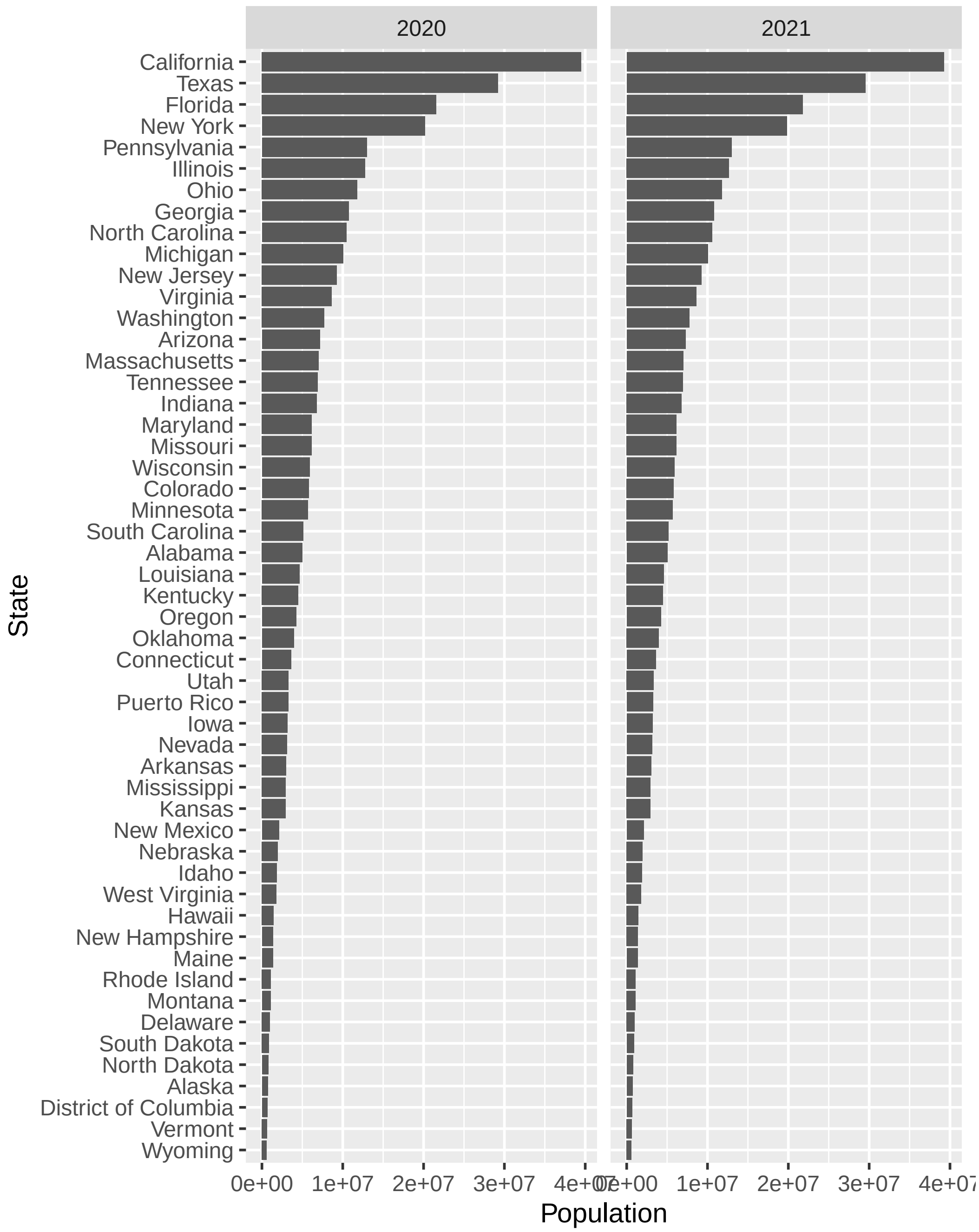
```
population <-resp_body_json(response, simplifyVector = TRUE)
```

6. Examine the `population` matrix you just created. Notice that 1) it is not tidy, 2) the column types are not what we want, and 3) the first row is a header. Convert `population` to a tidy dataset. Remove the state ID column and change the name of the column with state names to `state_name`. Add a column with state abbreviations called `state`. Make sure you assign the abbreviations for DC and PR correctly. Hint: Use the **janitor** package to make the first row the header.

```r
library(tidyverse)
library(janitor)
population <- population |> ## Use janitor row to names function
  as_tibble() |>
  row_to_names(row_number = 1) |>
  select(-state) |>
  pivot_longer(cols = c(POP_2020, POP_2021),
               names_to = "year",
               values_to = "population") |>
  mutate(year = as.numeric(str_remove(year, "POP_")),
         population = as.numeric(population), # parese all relevant colunns
          ↪  to numeric
         state_abb = state.abb[match(NAME, state.name)]
         ) |>
  rename(state_name = NAME) |>
  mutate(state_abb = case_when(state_name == "District of Columbia" ~ "DC",
                               state_name == "Puerto Rico" ~ "PR",
                               TRUE ~ state_abb)) # use case_when to add
                                ↪  abbreviations for DC and PR
```

7. As a check, make a barplot of states' 2021 and 2022 populations. Show the state names in the y-axis ordered by population size. Hint: You will need to use `reorder` and use `facet_wrap`.

```r
population |>
  mutate(state_name = reorder(state_name, population)) |> # reorder state
  ggplot(aes(x = state_name, y = population)) + # assign aesthetic mapping
  geom_col() + # use geom_col to plot barplot
  coord_flip() + # flip coordinates
  facet_wrap(~year) +# facet by year
  labs(
    x = "State",
    y = "Population"
  )
```



8. The following URL:

```
url <- "https://github.com/datasciencelabs/2025/raw/refs/heads/main/data/reg
↪ ions.json"
```

points to a JSON file that lists the states in the 10 Public Health Service (PHS) defined by CDC. We want to add these regions to the `population` dataset. To facilitate this create a data frame called `regions` that has two columns `state_name`, `region`, `region_name`. One of the regions has a long name. Change it to something shorter.

```
library(jsonlite)
library(purrr)
url <- "https://github.com/datasciencelabs/2025/raw/refs/heads/main/data/reg
↪ ions.json"
regions <- fromJSON(url, simplifyDataFrame = FALSE) # changes from DF to
↪ list
regions_df <- regions |>
  map_df(~ as_tibble(.x)) |> # convert list to data frame. You can use
   ↪ map_df in purrr package
  mutate(region_name = recode(region_name, # renaming long region to shorter
                             "New York and New Jersey, Puerto Rico, Virgin
                              ↪ Islands" = "other")) |>
  rename(state_name = states) # renaming col name for easier merging
```

9. Add a region and region name columns to the `population` data frame.

```
population <- left_join(population, regions_df, by = "state_name") # used
↪ left join to keep population columns
```

10. From reading https://data.cdc.gov/ we learn the endpoint `https://data.cdc.gov/resource/pwn4-m3yp.` provides state level data from SARS-COV2 cases. Use the **httr2** tools you have learned to download this into a data frame. Is all the data there? If not, comment on why.

```
api <- "https://data.cdc.gov/resource/pwn4-m3yp.json"
cases_raw <- request(api) |>
   req_perform() |>
   resp_body_json(simplifyDataFrame = TRUE)

nrow(cases_raw)
```

```
[1] 1000
```

No, not all the data is there because we downloaded the data with a default n = 1000. There are more rows, but we did not output it.

We see exactly 1,000 rows. We should be seeing over 52×3 rows per state.

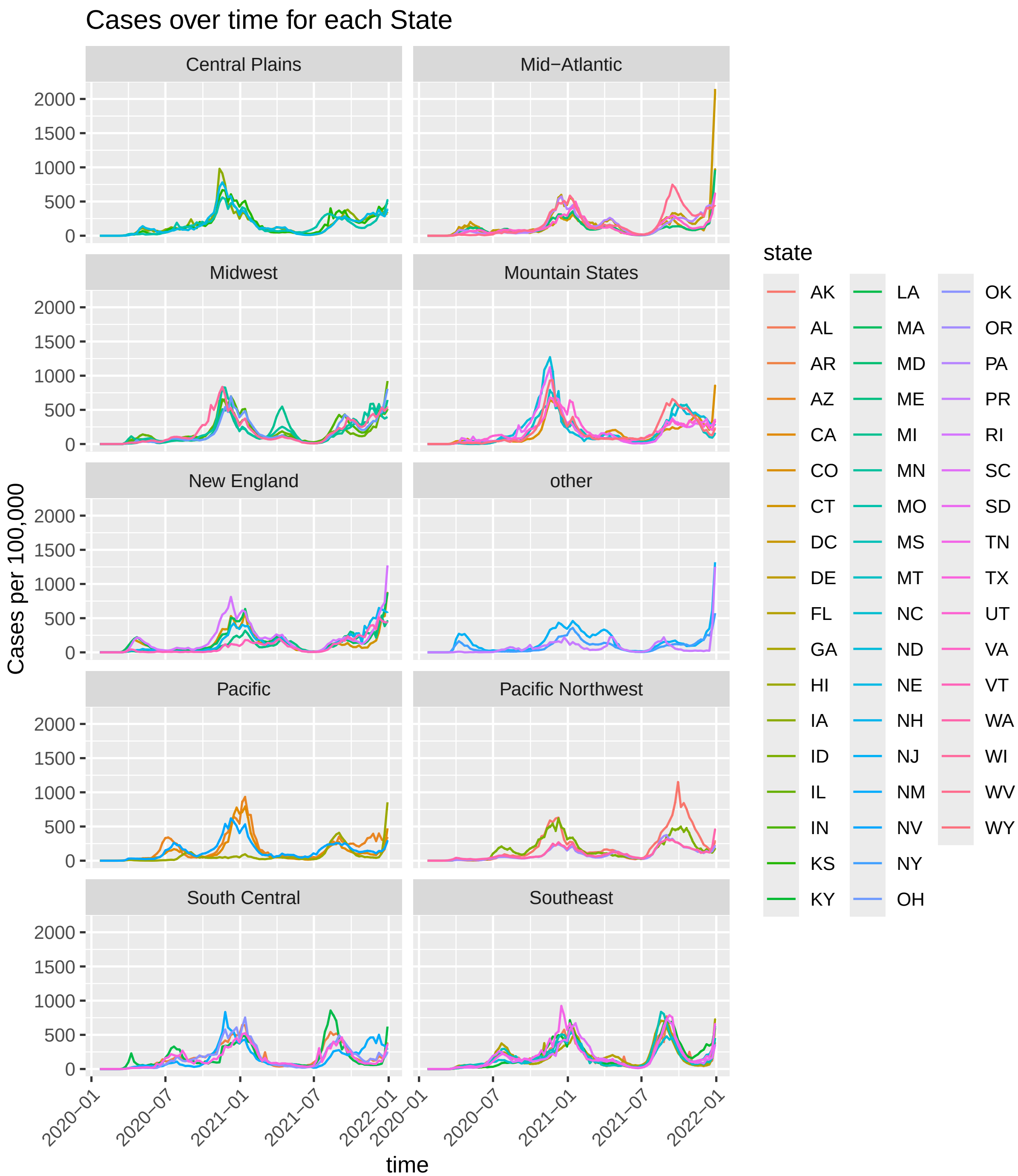
11. The reason you see exactly 1,000 rows is because CDC has a default limit. You can change this limit by adding `$limit=10000000000` to the request. Rewrite the previous request to ensure that you receive all the data. Then wrangle the resulting data frame to produce a data frame with columns `state`, `date` (should be the end date) and `cases`. Make sure the cases are numeric and the dates are in `Date` ISO-8601 format.

```
api <- "https://data.cdc.gov/resource/pwn4-m3yp.json"
cases_raw <- request(api) |>
  req_url_query(`$limit` = "10000000000") |> # increase limit
  req_perform()|> # perform a request
  resp_body_json(simplifyDataFrame = TRUE)  # we use the fromJSON function
   ↪ from the jsonlite package

cases_clean <- cases_raw |>
  mutate(date = as.Date(end_date), # changed date to ISO-8601 format
         cases = parse_number(new_cases)) |>  # made cases numeric
 select(state, date, cases) # only selected 3 columns
```

12. For 2020 and 2021, make a time series plot of cases per 100,000 versus time for each state. Stratify the plot by region name. Make sure to label you graph appropriately.

```
cases_clean |>
  rename(state_abb = state) |>  # renamed for ease of merging for regions
  left_join(population, by = "state_abb") |>
  rename(state = state_abb) |>
  filter(!is.na(region_name),
         date >= as.Date("2020-01-01") & date <= as.Date("2021-12-31")) |> #
↪ removing states not in regions
  mutate(cases_per_100 = 100000 * cases / population) |>  # making cases per
   ↪ 100000
  ggplot(aes(x=date, y = cases_per_100, col = state)) +
  geom_line() +
  labs(
    x = "time",
    y = "Cases per 100,000",
    title = "Cases over time for each State"
  ) +
  facet_wrap(~ region_name, ncol = 2) + # stratify by region name
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



13. The dates in the `cases` dataset are stored as character strings. Use the **lubridate** package to properly parse the `date` column, then create a summary table showing the total COVID-19 cases by month and year for 2020 and 2021. The table should have columns for year, month (as month name), and total cases across all states. Order by

year and month. Use the **knitr** package and `kable()` function to display the results as a formatted table.

I already turned the date into a date in the prior question

```
library(lubridate)
library(knitr)
cases_clean |>
  filter(date >= as.Date("2020-01-01") & date <= as.Date("2021-12-31")) |>
  mutate(
    Year = year(date),
    Month = lubridate::month(date, label = TRUE, abbr = FALSE),
    day = day(date)
  ) |>
  group_by(Year, Month) |>
  summarise("Total COVID-19 Cases" = sum(cases)) |>
  kable()
```

```
`summarise()` has grouped output by 'Year'. You can override using the
`.groups` argument.
```

Year	Month	Total COVID-19 Cases
2020	January	11
2020	February	68
2020	March	68245
2020	April	974032
2020	May	650943
2020	June	654904
2020	July	1989512
2020	August	1461283
2020	September	1415438
2020	October	1628598
2020	November	3932646
2020	December	7027128
2021	January	5808063
2021	February	2667511
2021	March	2068441
2021	April	1773591
2021	May	972915
2021	June	493635

Year	Month	Total COVID-19 Cases
2021	July	1137440
2021	August	3572562
2021	September	5027537
2021	October	2356302
2021	November	2322814
2021	December	5615644

14. The following URL provides additional COVID-19 data from the CDC in JSON format:

```
deaths_url <- "https://data.cdc.gov/resource/9bhg-hcku.json"
```

Use **httr2** to download COVID-19 death data from this endpoint. Make sure to remove the default limit to get all available data. Create a clean dataset called `deaths` with columns `state`, `date`, and `deaths` (renamed from the original column name). Ensure dates are in proper Date format and deaths are numeric.

```
# Your code here
covid_cases <- request(deaths_url) |>
  req_url_query(`$limit` = "10000000000") |> # increase limit
  req_perform()|> # perform a request
  resp_body_json(simplifyDataFrame = TRUE)  # we use the fromJSON function
   ↪  from the jsonlite package

covid_cases_clean <- covid_cases |>
  mutate(date = as.Date(end_date), # changed date to ISO-8601 format
         deaths = as.numeric(covid_19_deaths)) |>  # made cases numeric
 select(state, date, deaths) |> # only selected 3 columns
  filter(state %in% population$state_name, # removing UNITED STATES and NYC
         !is.na(deaths)) # removing death = NA
```

15. Using the `deaths` dataset you created, make a bar plot showing the total COVID-19 deaths by state. Show only the top 10 states with the highest death counts. Order the bars from highest to lowest and use appropriate labels and title.

```
# Your code here
covid_cases_clean |>
  group_by(state) |>
  summarise(total_case = sum(deaths)) |>
  arrange(desc(total_case)) |>
```

```r
head(n = 10) |>
ggplot(aes(x = reorder(state, total_case), y = total_case)) +
geom_col() +
coord_flip() +
labs(
  x = "Total COVID-19 deaths",
  y = "State",
  title = "Top 10 States with Highest COVID Death Counts"
)
```